Apprentissage non-paramétrique

Boi Faltings

Laboratoire d'Intelligence Artificielle
boi.faltings@epfl.ch
http://moodle.epfl.ch/

Modèles structurés

Souvent, le modèle à apprendre est trop complexe pour des modèles (paramétriques) simples.
Modèles structurés: diviser pour régner

- séparer les exemples par une classification, et ensuite
- apprendre des modèles simples (paramétriques) pour les sous-ensembles.

Types de modèles structurés

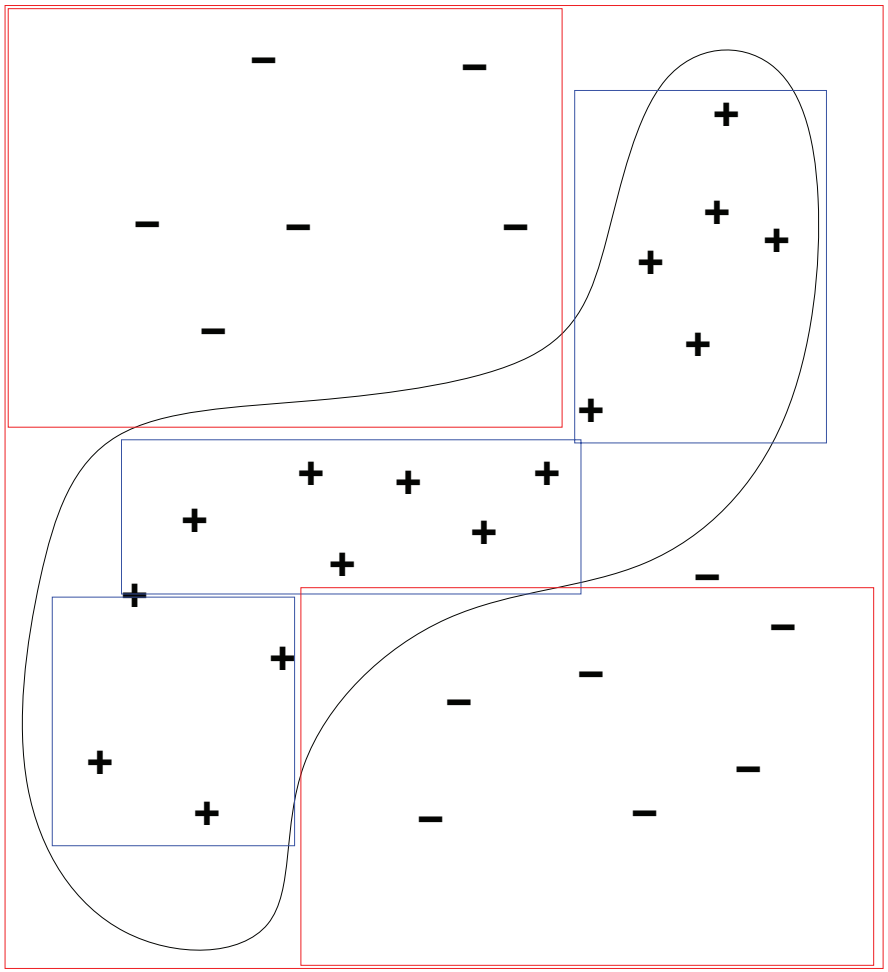
- Arbres de classification (ID3/C4.5)
- Combinaison pondérée (Boosting)

Modèles logiques disjonctifs

Apprentissage de classifications logiques = recherche d'une hypothèse $\mathtt{h}$ qui donne l'appartenance.
2 possibilités pour étendre à des modèles disjonctifs:

1. admettre des hypothèses $\mathtt{h}$ qui ne sont pas satisfaites par tous les exemples positifs.
   ⇒ classifications disjonctives
2. admettre des hypothèses $\mathtt{h}$ qui sont satisfaites par certains exemples négatifs.
   ⇒ listes de décision

Types de modèles



= disjonction

= liste de decisions

Arbres de classification

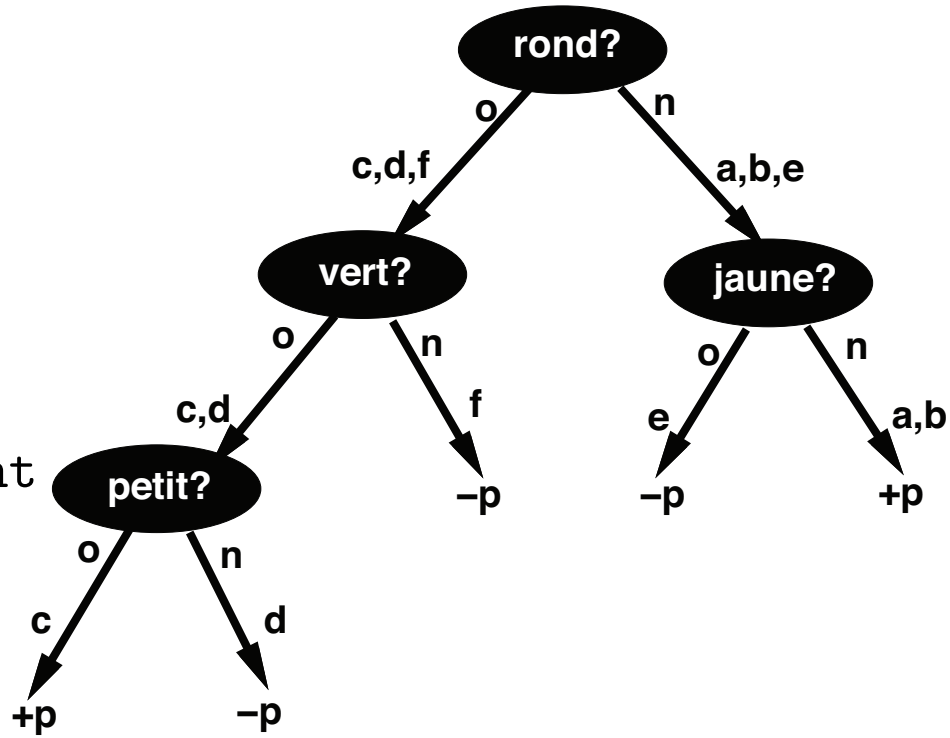
Arbre de classification = structure qui sert à déterminer la classe d'un exemple X:

- chaque noeud spécifie un prédicat P. Si $P(X)$ est vrai, on passe au successeur gauche, sinon, au successeur droit.
- les noeuds terminaux désignent les classes.
- partir du noeud initial.

Represente une décomposition disjonctive de l'espace d'exemples.

Arbre de classification

a) grand,allongé,rouge,piquant
b) petit,allongé,rouge,piquant
c) petit,rond,vert,piquant
d) grand,rond,vert,¬piquant
e) petit,allongé,jaune,¬piquant
f) petit,rond,rouge,¬piquant



Apprendre des arbres de classification: ID3

- But de ID3: étant donné des exemples appartenant à k classes différentes, construire l'arbre de classification optimal qui permet de distinguer les classes.
- "optimal" = profondeur moyenne minimale
- ID3 utilise tous les exemples en même temps, et construit l'arbre de décision de manière incrémentale.
- Pas élémentaire de la procédure: sélection de l'attribut à choisir pour le noeud courant: doit laisser aussi peu d'incertitude que possible.

Function ID3(E)
if E = {} **then**
  return NIL
else
  if ∀ e ∈ E classe(e)=c **then**
    return c
  else
    P ← attribut ∈A qui réduit au plus l'entropie de la classe
    L ← $\{e|e \in E$ et P(e) = succès$\}$
    R ← $\{e|e \in E$ et P(e) = echec $\}$
    N ← noeud vide, N.P ← P
    N.left ← ID3(L), N.right ← ID3(R)
    return N

Choisir le meilleur attribut

L'attribut doit être incertain:

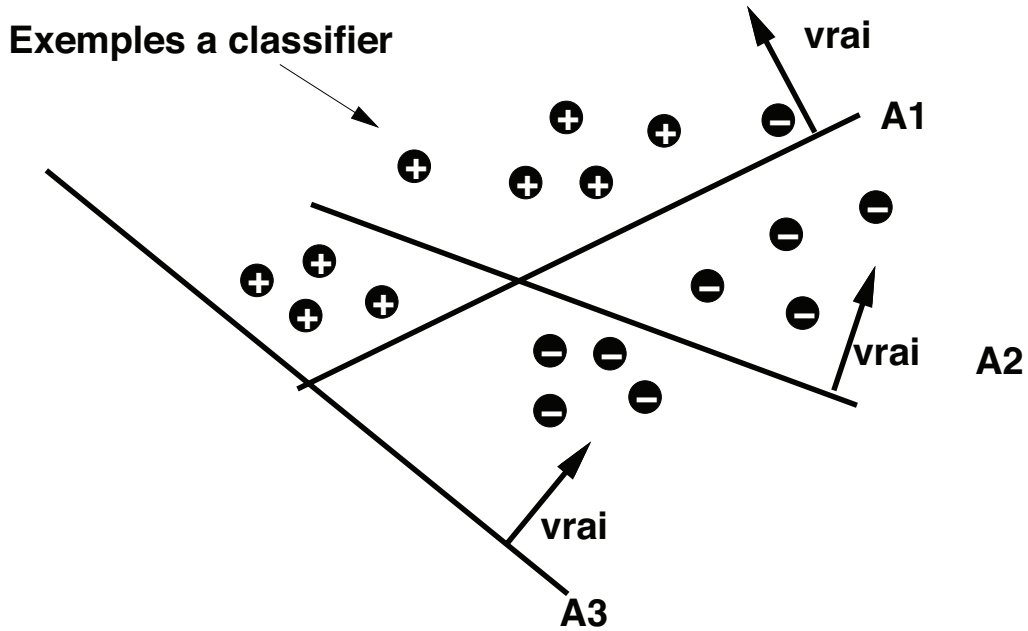
A1,A2 vrai 10 sur 18 fois
très incertain

A3 vrai 18 sur 18
aucune information

L'attribut doit être corrélé à la classe:

A1: vrai: $p(+) = 9/10$
faux: $p(+) = 0$
utile

A2: vrai: $p(+) = 5/10$
faux: $p(+) = 4/8$
inutile



Formaliser l'incertitude par l'entropie

Entropie = l'incertitude de la valeur d'une variable aléatoire X à valeurs $v_1, v_2, ..., v_k$:

$$H(X) = -\sum_{i=1}^{k} p(X = v_i) \cdot log_2 p(X = v_i) \text{ bit}$$

Entropie de X étant donné la valeur d'une autre variable $Y = w_j$:

$$H(X|Y = w_j) = -\sum_{i=1}^{k} p(X = v_i|Y = w_j) \cdot log_2 p(X = v_i|Y = w_j)$$

$\Rightarrow$ incertitude moyenne si la valeur de $Y \in \{w_1, .., w_m\}$ est connue:

$$H(X|Y) = \sum_{j=1}^{m} p(Y = w_j) \cdot H(X|Y = w_j)$$

Choix du meilleur attribut

C = classe d'un exemple e parmi k classes
Y = valeur d'un attribut A parmi m valeurs
⇒ incertitude sur la classe de X, étant donné que $A(X) = a_j$:

$$H(C|a_j) = -\sum_{i=1}^{k} p(c_i|a_j) \cdot log_2 p(c_i|a_j)$$

⇒ incertitude moyenne sur classe(X), valeur de A connue:

$$H(C|A) = -\sum_{j=1}^{m} p(a_j) \cdot \sum_{i=1}^{k} p(c_i|a_j) \cdot log_2 p(c_i|a_j)$$

⇒ choisir l'attribut A qui donne la plus petite incertitude (entropie) restante
⇒ la valeur la moins élevée de $H(C|A)$

Exemple...

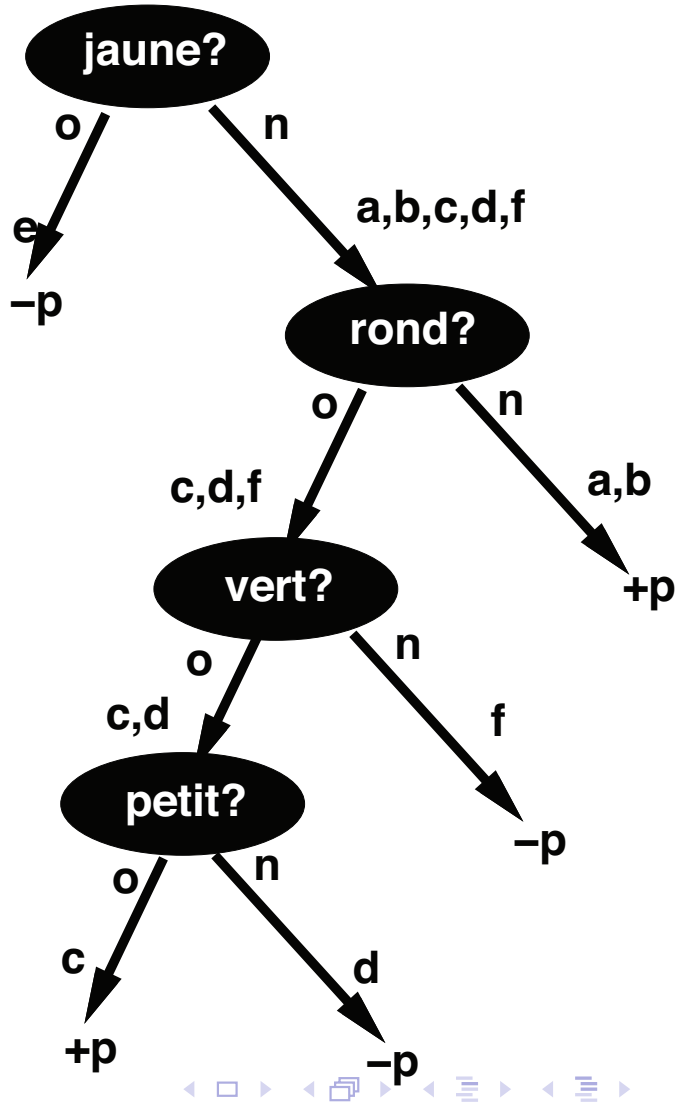
a) `grand,allongé,rouge,piquant`
b) `petit,allongé,rouge,piquant`
c) `petit,rond,vert,piquant`
d) `grand,rond,vert,¬piquant`
e)`petit,allongé,jaune,¬piquant`
f) `petit,rond,rouge,¬piquant`

Entropie de `rouge`:

$Pr(v) = 0.5$ (a,b,f), $Pr(f) = 0.5$ (c,d,e)
$Pr(piquant|v) = 0.66$, $Pr(\neg\ piquant|v) = 0.33$
$Pr(piquant|f) = 0.33$, $Pr(\neg\ piquant|f) = 0.66$
$H(C|v) = -\ 0.66\ log_2(0.66) - 0.33\ log_2(0.33) = 0.92$ bit
$H(C|f) = -\ 0.33\ log_2(0.33) - 0.66\ log_2(0.66) = 0.92$ bit
moyenne $H(C|A) = 0.92$ bit

Construction de l'arbre

1. (a,b,c,d,e,f)
   `grand,petit,vert`: 1 bit
   `rond,allongé,rouge`: 0.92 bit
   `jaune`: 0.81 bit
   ⇒ choisir `jaune`
2. (a,b,c,d,f)
   `grand,petit`: 0.955 bit
   `rond,allongé`: 0.554 bit
   `rouge,vert`: 0.955 bit
   ⇒ choisir `rond`
3. (c,d,f)
   `grand,petit`: 0.66 bit
   `rouge,vert`: 0.66 bit
   ⇒ choisir `vert`
4. (c,d) `grand,petit`: 0 bit ⇒ `petit`



Valeurs multiples

- Le critère d'entropie privilégie des attributs ayant beaucoup de valeurs.
- Par exemple, un numéro d'identification réduira l'incertitude à zéro, mais donne un arbre peu intéressant.
- Compenser en divisant la réduction d'incertitude par le nombre de branches générées.

$\Rightarrow$ choisir A pour maximiser:

$$\frac{H(C) - H(C|A)}{|valeurs(A)|}$$

Autres critères pour la division

But de la division: réduire la diversité des sous-classes qui résultent
⇒ l'entropie est la mesure idéale, mais on peut en utiliser d'autres:

- variance des sous-classes (si la classe c est une valeur numérique):

$$1/n \sum_{i=1}^{n} (c_i - \overline{c})^2$$

- fraction d'exemples n'appartenant pas à la classe la plus fréquente:

$$1/n |\{c_i | c_i \neq \text{classe la plus fréquente}\}|$$

Utilisation pour la regression

- Regression = prévision d'une valeur numérique.
- Peut se faire par arbre de décision en remplacant la classe par la valeur numérique à prédire.
- L'entropie comme critère de division s'applique de façon identique.
- Arrèter la division quand les valeurs sont suffisements uniformes.
- Prédiction = moyenne des exemples (ou modèle paramétrique).

Extensions

ID3 est un algorithme très efficace et beaucoup utilisé en pratique, mais:

- certaines décisions peuvent être un hasard de la sélection des exemples.
- la classification n'est pas toujours bonne pour de nouveaux exemples.
- les arbres de classification qui en résultent sont souvent difficiles à interpréter.

Extensions:

- élaguer l'arbre
- transformation en règles

Le problème du surapprentissage (overfitting)

- L'utilité de l'arbre ne dépend pas des exemples donnés, mais de sa performance sur des *nouveaux* exemples.
- Les parties proches des feuilles sont souvent construites sur la base de très peu d'exemples et sont un résultat aléatoire de la sélection des exemples.
- L'arbre donne alors de très mauvais résultats pour de nouveaux exemples.
- Comment estimer la qualité:
  a) garder une partie des exemples en réserve pour évaluer le taux d'erreur de la classification
  b) utiliser des méthodes statistiques (PAC)

PAC: Chiffrer la qualité

- **P**roblement:
  la classification n'est pas approximativement correcte avec probabilité δ
- **A**pproximativement:
  la probabilité d'erreur de classification est inférieure à ϵ.
- **C**orrect

En supposant que

- on donne N exemples
- dont la distribution correspond à la réalité
- pour apprendre un concept parmi $|H|$ possibilités
- l'algorithme rend un résultat correct sur tous les exemples

$\Rightarrow$ on peut chiffrer la relation entre N, $|H|$, δ et ϵ!

Estimer la qualité...

- Supposons que $h \in H$ a un taux d'erreur $> \epsilon$
  $\Rightarrow$ probabilité que h soit correcte sur les N exemples

$$Pr(correct(h, N)) < (1 - \epsilon)^N$$

- Probabilité que parmi les $|H|$ possibilités, il y a un h qui est correcte sur N exemples mais avec un taux d'erreur $> \epsilon$ doit être $< \delta$:

$$\delta > |H| \cdot (1 - \epsilon)^N$$

  $\Rightarrow$ pour garantir une borne δ, il faut que N respecte:

$$N \geq \frac{\log(\delta/|H|)}{\log(1 - \epsilon)}$$

Estimer la quantité d'exemples

Autre point de vue: estimer le nombre d'exemples qu'il faut pour obtenir la qualité voulue:

- Si $|H|$ est en croissance exponentielle avec le nombre d'attributs
  $\Rightarrow$ croissance linéaire du nombre d'exemples requis.
- Souvent, la croissance est plus qu'exponentielle (par exemple, arbres de classification)
  $\Rightarrow$ forte croissance du nombre d'exemples, même pour des modèles qui paraissent simples.

Exemple...

Induction d'un arbre de décision:

- 10 attributs
- $\delta < 0.01$
- $\epsilon < 0.001$

11 choix pour le premier noeud (10 attributs + arbre vide)
fois 10 choix pour chacun des deux 2e noeuds
fois 9 choix pour chacun des quatre 3e noeuds

$$|H| = 11 \cdot 10^2 \cdot 9^4 \cdot 8^8 \ldots \cdot 2^{2^9} \approx 2.6579 \cdot 10^{35}$$

arbres, donc il faut $N \geq 86'131$ exemples pour apprendre ce concept!
Si $\epsilon = 0.05$, $N \geq 1'680$ exemples.

Elaguer l'arbre...

- Les décisions introduites vers les feuilles de l'arbre sont basées sur très peu d'exemples.
- Estimer le taux d'erreurs et élaguer le sous-arbre s'il est trop élevé.
- L'arbre taillé sera alors remplacé par une seule feuille dont la classe est celle qui était la plus fréquente.

Critères pour l'élagage

Déterminer le nombre d'erreurs probable d'un sous-arbre:

- utiliser un nouveau jeu d'exemples.
- hypothèse d'indépendance des erreurs ⇒ statistique permet d'estimer la fréquence d'erreurs de l'arbre (comme le PAC).

Elaguer quand le nombre d'erreurs introduit par l'élagage est inférieur au nombre d'erreurs sans taillage.

Forêts d'arbres de décision (Random Forests)

- Construire une multitude d'arbres de décision:
  - varier le choix de l'attribut pour la division. (y.c. choix sous-optimal).
  - apprendre des arbres sur différents sous-ensembles d'exemples d'apprentissage (bagging).
- Combinaison par décision majoritaire: la classification selectionné par le plus grand nombre d'arbres.
- Reduit le problème du surapprentissage.

Arbre ⇒ Règles

- Les décisions de l'arbre sont toujours liées à une séquence de parcours.
- Souvent, il serait plus pratique d'avoir des règles valables partout:

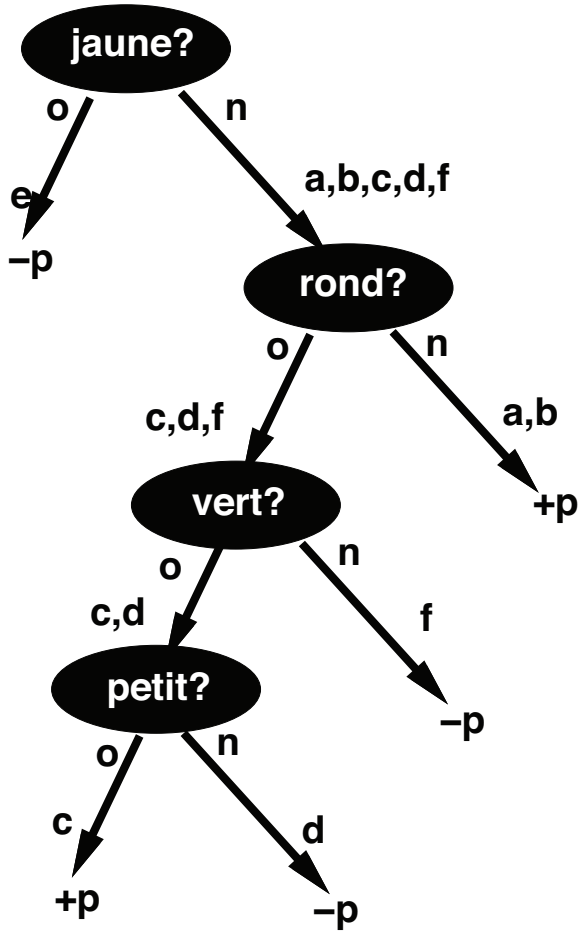
$$\texttt{jaune} \Rightarrow \neg\, \texttt{piquant}$$
$$\neg\, \texttt{jaune} \wedge \neg\, \texttt{rond} \Rightarrow \texttt{piquant}$$
$$\neg\, \texttt{jaune} \wedge \texttt{rond} \wedge \neg\, \texttt{vert} \Rightarrow \neg\texttt{piquant}$$
$$....$$

- Permet l'utilisation du modèle dans un système à base de connaissances.

Génération des règles

Chaque parcours de l'arbre définit une règle:

1. jaune ⇒ ¬ piquant
2. ¬jaune∧¬rond ⇒ piquant
3. ¬jaune∧rond∧¬vert ⇒ ¬piquant
4. ¬jaune∧rond∧vert∧petit ⇒piquant
5. ¬jaune∧rond∧vert∧¬petit ⇒ ¬piquant



Génération des règles

Chaque parcours de l'arbre définit une règle:

1. jaune ⇒ ¬ piquant
2. ¬jaune∧¬rond ⇒ piquant
3. ¬jaune∧rond∧¬vert ⇒ ¬piquant
4. ¬jaune∧rond∧vert∧petit ⇒piquant
5. ¬jaune∧rond∧vert∧¬petit ⇒ ¬piquant

Certaines règles peuvent être simplifiées:

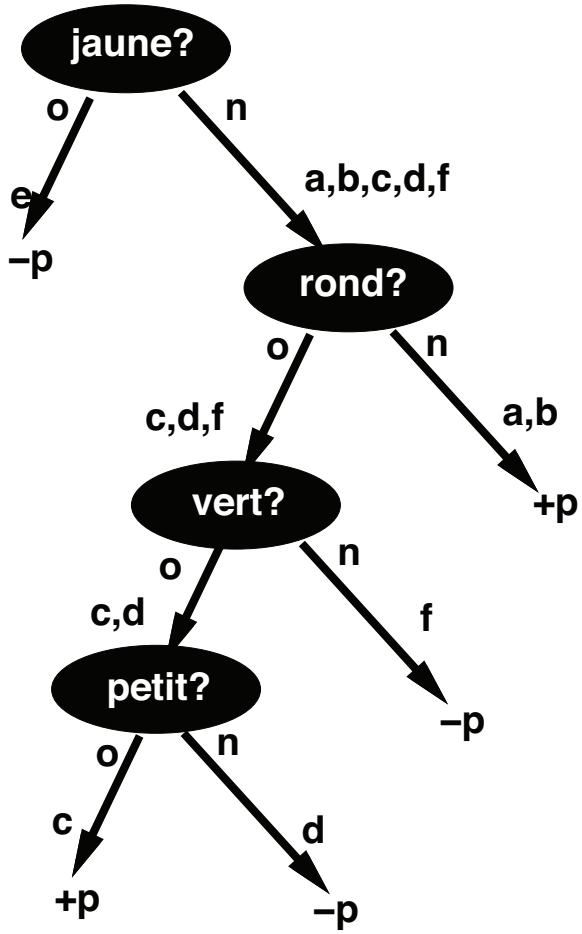
3' rond ∧ ¬ vert ⇒ ¬ piquant

4' rond ∧ vert ∧ petit ⇒ piquant

4'' vert ∧ petit ⇒ piquant

5' rond ∧ vert ∧ ¬ petit ⇒ ¬ piquant

5'' vert ∧ ¬ petit ⇒ ¬ piquant



Trouver les simplifications

- Simplification d'une règle à la fois.
- Laisser tomber un prédicat à la fois et évaluer le résultat sur l'ensemble des exemples.
- Il peut y avoir plusieurs manières de simplifier (4' et 4").
- Parfois, on devrait tolérer que certains exemples ne sont plus couverts ou couverts incorrectement:
  la simplification élimine du bruit.

Boosting

- ID3/C4.5 utilise différents critères en fonction du parcours de l'exemple à classifier.
- On peut aussi imaginer une structure "plate" qui applique plusieurs critères en parallèle.
- Le boosting permet d'apprendre une combinaison de modèles *faibles* pour obtenir une plus grande précision.
- Modèle faible: taux d'erreur $< 50\%$.
- Combinaison = vote ponderé.

Méthode faible d'induction

- Entrées:

  n examples $(\underline{x}_i, c_i), i \in 1..n;\ c_i \in \{0, 1\}$
  distribution de probabilités $p_i, \sum_{i=1}^{n} p_i = 1$

- Apprend:

  classificateur $h(\underline{x})$ *avec probabilité d'erreur* $< 50\%$*:*
  $\epsilon = \sum_{i=1}^{n} p_i |h(\underline{x}_i) - c_i| < 0.5$

  pour une distribution donnée de probabilités p_i!

- On peut renverser le résultat $(1 - h(\underline{x}))$ si nécessaire.

Adaboost

Initialiser: $w_i \leftarrow 1/n, i = 1..n$

Algorithme = Iteration pour $t \in 1..k$

1. for $i \in 1..n, p_i \leftarrow \frac{w_i}{\sum_{i=1}^{n} w_i}$
2. $h_t \leftarrow$ résultat de la méthode d'induction faible avec les exemples et probabilités p_i
3. taux d'erreurs $\epsilon \leftarrow \sum_{i=1}^{n} p_i |h(\underline{x}_i) - c_i|$
4. $\beta_t \leftarrow \frac{\epsilon}{(1-\epsilon)}$
5. for $i \in 1..n, w_i \leftarrow w_i \beta_t^{1-|h(\underline{x}_i)-c_i|}$

Résultat:

$$h_f(\underline{x}) = \begin{cases} 1 & si \sum_{t=1}^{k} (-log \beta_t)(h_t(\underline{x}) - 1/2) \geq 0 \\ 0 & autrement \end{cases}$$

Exemple: piments

a) `grand,allongé,rouge,piquant`
b) `petit,allongé,rouge,piquant`
c) `petit,rond,vert,piquant`
d) `grand,rond,jaune,¬piquant`
e) `petit,allongé,jaune,¬piquant`
f) `grand,rond,rouge,¬piquant`

Méthode faible: trouver un attribut/valeur qui prédit `piquant`.

Exemple: piments

h_1: `allongé` $\Rightarrow$ `piquant`

erreurs	ϵ_1	β_1	w_1	w_2	w_3	w_4	w_5	w_6
c,e	$1/3$	$1/2$	$1/12$	$1/12$	$1/6$	$1/12$	$1/6$	$1/12$

$P = (1/8, 1/8, 1/4, 1/8, 1/4, 1/8)$

h_2: `jaune` $\Rightarrow \neg$ `piquant`

erreurs	ϵ_2	β_2	w_1	w_2	w_3	w_4	w_5	w_6
f	$1/8$	$1/7$	$1/84$	$1/84$	$1/42$	$1/84$	$1/42$	$1/12$

$P = (1/14, 1/14, 1/7, 1/14, 1/7, 1/2)$

h_3: `vert` $\Rightarrow$ `piquant`

erreurs	ϵ_3	β_3	w_1	w_2	w_3	w_4	w_5	w_6
a,b	$1/7$	$1/6$	$1/84$	$1/84$	$1/252$	$1/504$	$1/252$	$1/84$

Exemple: piments

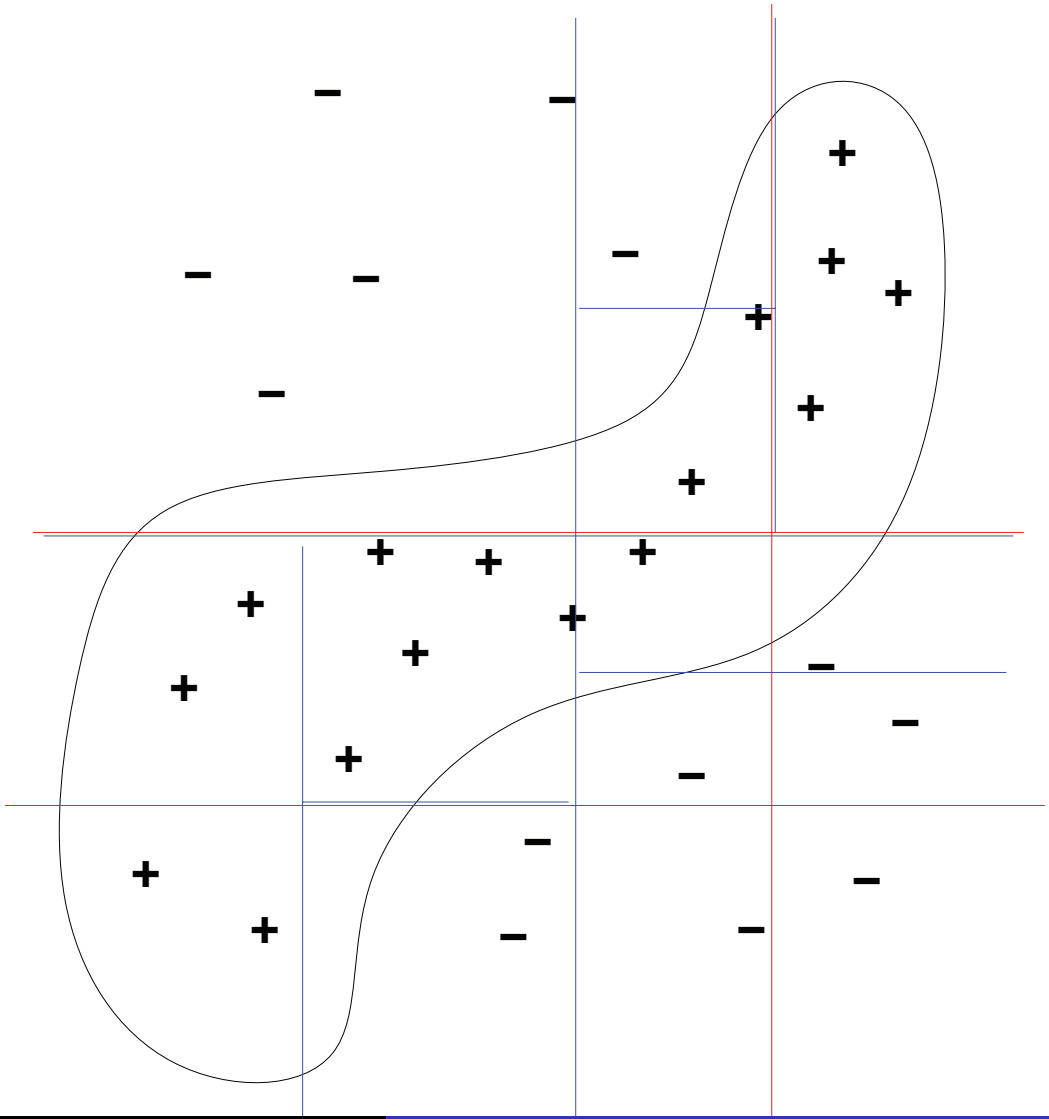
Résultat:

$$\begin{aligned} & h_f(\underline{x}) \\ = \; & log(2)h_1(\underline{x}) + log(7)h_2(\underline{x}) + log(6)h_3(\underline{x}) - \frac{log(2) + log(7) + log(6)}{2} \\ = \; & 0.69h_1(\underline{x}) + 1.95h_2(\underline{x}) + 1.8h_3(\underline{x}) - 2.22 \end{aligned}$$

Performance:

Exemple	h_1	h_2	h_3	h_f
a) grand,allongé,rouge,piquant	1	1	0	0.42
b) petit,allongé,rouge,piquant	1	1	0	0.42
c) petit,rond,vert,piquant	0	1	1	1.53
d) grand,rond,jaune,¬piquant	0	0	0	-2.22
e) petit,allongé,jaune,¬piquant	1	0	0	-1.53
f) grand,rond,rouge,¬piquant	0	1	0	-0.27

Arbres de décision et boosting



arbre de decision

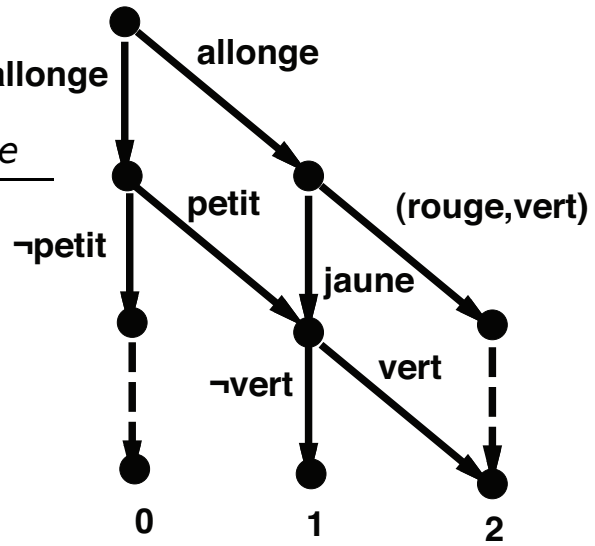
boosting

Martingale boosting

Faire dépendre le prochain classificateur des résultats des classificateurs précédents:

Décision par majorité: ≥ 2

Exemple	h_1	h_2	h_3	*score*
a) `grand,allongé,rouge,piquant`	1	1	-	2
b) `petit,allongé,rouge,piquant`	1	1	-	2
c) `petit,rond,vert,piquant`	0	1	1	2
d) `grand,rond,jaune,¬piquant`	0	0	-	0
e) `petit,allongé,jaune,¬piquant`	1	0	0	1
f) `grand,rond,rouge,¬piquant`	0	0	-	0



Application: Prédiction de pannes de réseaux électriques

- Réseau de distribution d'électricité tombe en panne à cause de vieux câbles.
- Tester les câbles les rend souvent inutilisables.
- En 2005, Consolidated Edison (New York) a introduit un système d'apprentissage pour prédire les câbles qui posent problème.
- Technique utilisée: Martingale boosting.
- Le système a économisé des dizaines de millions de dollars/année en coûts de maintenance inutile.

Résumé

- Apprentissage de concepts disjonctifs
- Apprentissage d'arbres de décision
- Taillage de l'arbre pour éliminer le surapprentissage
- Traduction arbre ⇒ règles
- Combinaison de méthodes faibles par boosting